Agent-Based Decision Making in Competitive Resource Environments: A Catan Study

Carlos Eckert

May 28, 2026

Abstract

Settlers of Catan presents a rich environment for studying competitive resource allocation under spatial constraint, stochastic production, and multi-agent rivalry. These properties are shared by real-world systems such as supply chain competition, urban land development, and market entry strategy. This paper investigates how agents with fundamentally different decision-making architectures perform within this environment, using Catan as a controlled proxy for broader questions about rational behavior under uncertainty. We implement four agents of increasing sophistication: a uniform-random baseline, a greedy heuristic agent with utility functions for each possible move, a flat Monte Carlo agent using law-of-large-numbers averaging over random rollouts, and a Monte Carlo Tree Search agent using UCB1 budget allocation over heuristic-guided and random rollouts. All agents are evaluated in round-robin 1v1 tournaments and a four-player all-agent tournament across hundreds of simulated games. Performance is measured by win rate and mean final victory points across both competition formats. Results reveal a consistent performance hierarchy across agent architectures in 1v1 play, while the four-player format produces a notably different competitive landscape, suggesting that decision-making complexity interacts with the number of competing agents in non-trivial ways. More broadly, the agents represent three distinct computational philosophies: rule-following, simulation, and adaptive exploration. Their corresponding performances offer insight into how different modes of strategic reasoning succeed and fail in competitive resource environments.

1 Introduction

Competition for scarce resources is a fundamental feature of many real-world systems. Firms race to establish distribution infrastructure near high-demand regions, developers bid for land parcels under spatial constraints, and market entrants allocate finite capital across resource types while responding to rival positioning. In each case, the underlying structure is the same: multiple agents compete for limited resources in an environment where outcomes are uncertain, and the actions of others directly affect what is available.

Settlers of Catan [Kla24] is a multiplayer board game in which players collect resources, build settlements and cities across a hex-grid map, and race to accumulate ten victory points. Its mechanics naturally encode the structure of competitive resource allocation: production is stochastically generated, determined by dice rolls whose outcomes follow a known probability distribution; the board is spatial, with placement decisions subject to adjacency and distance constraints; and competition is direct, as players occupy the same finite set of vertices and can disrupt each other through the robber mechanic. These properties make Catan a compact and tractable environment for studying multi-agent competition under uncertainty. Beyond its structural appeal, this project originated from an interest in the game of Catan and a curiosity about how automated agents would navigate decisions that human players sometimes struggle with.

Most formal comparisons of agent architectures are conducted in perfect-information, deterministic environments such as chess or Go. Catan departs from this setting in two important ways: resource production is governed by dice rolls that introduce an element of randomness, and the spatial board creates asymmetric positional advantages that persist throughout the game. Świechowski et al. [SPMK15] survey agent architectures across general game-playing environments and identify heuristic evaluation and Monte Carlo methods as the two dominant paradigms, noting that neither universally

dominates the other across game types. Auer, Cesa-Bianchi, and Fischer [ACBF02] establish the theoretical foundation for Upper Confidence Bound algorithms, proving that UCB1 achieves logarithmic regret in the multi-armed bandit setting, a result that motivates its use in action selection within our Monte Carlo Tree Search agent.

This paper implements four agents of increasing decision-making sophistication in a full Catan simulation and evaluates their performance across 1v1 and four-player tournament formats. The agents represent three modes of calculation: rule-following, in which utility functions score and select actions based on domain knowledge; simulation, in which candidate actions are evaluated by averaging outcomes over randomly played futures; and adaptive exploration, in which a bandit algorithm allocates simulation budget toward promising actions while maintaining exploration guarantees. By comparing these agents in a tournament setting, this paper asks: when and why does rule-based calculation outperform search?

2 Model and Methods

2.1 Game Environment and Board Representation

Settlers of Catan is played on a hex-grid board consisting of nineteen resource-producing hexes arranged in a fixed geometric pattern, surrounded by ocean tiles. Each interior hex produces one of five resource types: lumber, brick, wool, grain, and ore. A desert hex produces nothing and serves as the initial placement site for the robber. Vertices sit at the intersections of hexes and serve as settlement and city placement sites; edges connect adjacent vertices and serve as road placement sites. The board is qualitatively characterized by its spatial scarcity: the number of valid placement sites is finite, placement is subject to a distance rule requiring that no two settlements occupy adjacent vertices, and the value of any given site depends on the dice numbers of its adjacent hexes.

Each hex is assigned a number token between two and twelve, excluding seven. On each turn, two six-sided dice are rolled and all hexes matching the result produce one resource card for each adjacent settlement and two for each adjacent city. The sum of two fair dice follows a symmetric triangular distribution over the integers two through twelve, with seven having the highest probability of 6/36. This distribution governs expected resource income and is central to all agent scoring functions described in subsequent subsections. Rolling a seven activates the robber, which blocks production on a target hex and allows the active player to steal one resource from an opponent with a settlement adjacent to that hex.

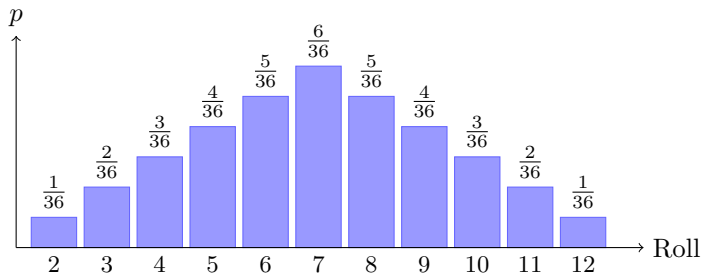

Figure 1: Probability distribution of the sum of two fair six-sided dice. The distribution is symmetric and triangular with mode at seven, $P(7) = 6/36$.

The full standard Catan ruleset is implemented, including resource production, settlement and city construction, road building, development cards, ports, longest road, and largest army. The only mechanic omitted relative to the full Catan game is agent-to-agent trading; all resource acquisition beyond dice production occurs through fixed-rate maritime trades with the bank.

As visualized below, the simulation is implemented entirely in MATLAB. A live visualization renders the hex board, current settlement and road placements, resource counts, victory point totals, and game feed after every action. This allows the full game arc to be observed in real time, including placement decisions, resource accumulation, and building sequences. The visualization was used to verify agent behavior qualitatively before running large-scale tournaments.

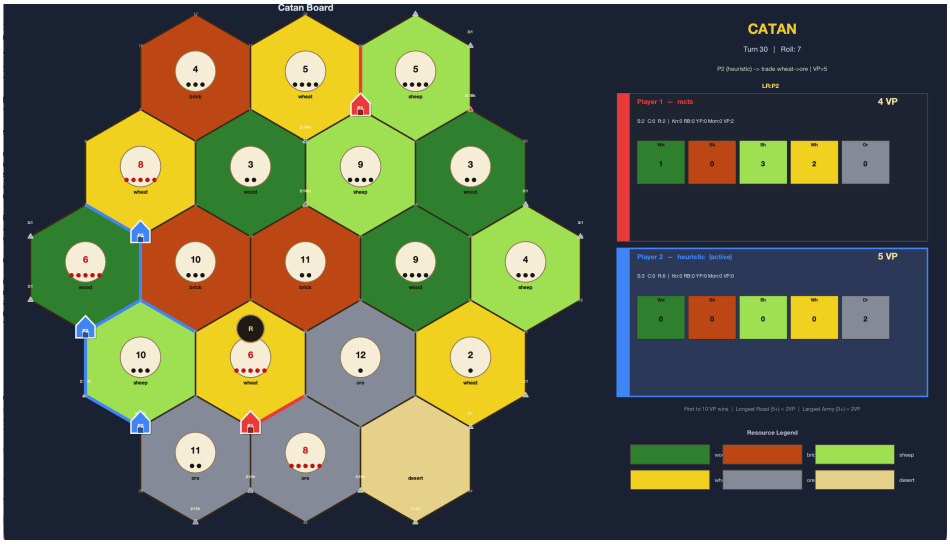

Figure 2: Catan as is implemented in MATLAB. Each player has settlements, a panel for their resources and victory points, and an overview of the game state.

2.2 Random Agent

The random agent serves as a baseline and provides a lower bound on performance against which all other agents are measured. It holds no model of the game state, performs no evaluation, and maintains no memory across turns. At each decision, it selects uniformly at random from the set of legal actions available in the current state:

$$a^* \sim \mathcal{U}\big(A(s)\big)$$

where $A(s)$ denotes the set of legal actions in state s.

The random agent is expected to win approximately $1/n$ of games in an n-player tournament against identically random opponents. Any deviation from this expectation in the tournament results reflects the contribution of the competing agent's decision-making strategy.

2.3 Heuristic Agent

The heuristic agent scores every legal action using a utility function and selects the action with the highest score:

$$a^* = \max_{a \in A(s)} \text{Score}(s, a)$$

This greedy approach does not require forward-looking calculations. The scoring function is decomposed by action type, with each sub-function providing an expectation for a given action in the current state.

2.3.1 Settlement Scoring

The settlement scoring function evaluates the expected utility of a candidate vertex v along four dimensions: expected resource production, resource need, diversity, and opponent blocking, with a penalty for robber proximity:

$$s(v) = w_p \sum_{h \in N(v)} p_h + w_n \sum_{h \in N(v)} p_h \cdot \max(0,\, d_r - x_r) + w_d \cdot T_{\text{new}} + w_b \cdot B(v)$$

where p_h is the dice probability of hex h, $d_r - x_r$ is the unmet build demand for resource r, T_{new} is the set of resource types not yet covered by existing settlements, and $B(v)$ is the set of adjacent vertices owned by opponents. The production term rewards high-pip placements; the need term weighs production

by how much the player currently lacks that resource; and the diversity term rewards coverage of new resource types. Default weights are given in Table 1.

Parameter	Default Value	Role
w_p	3.0	Expected production weight
w_n	1.5	Resource need weight
w_d	1.0	Diversity weight
w_b	0.2	Blocking weight
w_r	1.8	Road expansion weight
w_c	2.5	City marginal utility weight

Table 1: Default heuristic agent weight parameters.

2.3.2 Road Scoring

The road scoring function evaluates a candidate edge e by the total production value of new settlement spots made reachable by placing a road there. Let V^+ be the set of vertices newly accessible after placing the road:

$$s(e) = \begin{cases} w_r \sum_{v \in V^+} \sum_{h \in N(v)} p_h & \text{if } V^+ \neq \emptyset \\ 0.40 + \mathbf{1}[\ell < 5 \leq \ell'] & \text{otherwise} \end{cases}$$

where the indicator term gives a bonus for crossing the longest road threshold of five. In theory, the threshold should move based on the road count of the user currently possessing longest road.

2.3.3 City Scoring

The city scoring function rewards upgrading a settlement at vertex v by the probability-weighted marginal utility of doubling production there, incorporating a diminishing returns factor on surplus resources:

$$s(v) = w_c \sum_{h \in N(v)} p_h \cdot \max\left(0.5,\ \frac{d_r^{\text{tot}}}{\max(1,\ x_r)}\right)$$

where d_r^{tot} is the total build demand for resource r across all building types and x_r is the current stock. The $1/x_r$ term captures diminishing returns: a resource already held in surplus contributes less marginal value than one the agent is short on.

2.3.4 Development Card Scoring

The development card scoring function uses a piecewise base value adjusted by three state-dependent signals:

$$s = 1.0 + 0.6 \cdot \mathbf{1}[\text{no build} \wedge \Sigma x \geq 4] + 0.8 \cdot \mathbf{1}[\text{VP} \geq w - 3]$$

where the first indicator fires when the agent has resources but no available build, and the second fires when the agent is within three victory points of winning.

2.3.5 Trade Scoring

The trade scoring function prioritizes build completion. Given a trade exchanging a given resource for the resource rec at rate r, the score is:

$$s = \begin{cases} \sum_b v_b - (r-2) \cdot 0.3 & \text{if trade unlocks a build} \\ 0.5\,\delta_{\text{rec}} - 0.2(r-2) + 0.05\,\sigma & \text{otherwise} \end{cases}$$

where v_b is the value of build b unlocked, δ_{rec} is unmet need for the received resource, and σ is surplus remaining after trading. The trade rate penalty $(r-2)\cdot 0.2$ discourages expensive bank trades when no build is immediately enabled.

2.3.6 Robber Scoring

The robber scoring function balances opponent disruption against own-hex penalty:

$$s(h) = p_h \cdot n_{\text{opp}} \cdot 1.2 + 0.12\,|x_{\text{target}}| \cdot 1.3^{\mathbf{1}[\text{target is leader}]} - p_h \cdot n_{\text{own}} \cdot 1.5$$

where n_{opp} and n_{own} are the number of opponent and own buildings on hex h, x_{target} is the target player's resource count, and the leader bonus scales steal value upward when targeting the current victory point leader.

2.4 Monte Carlo Agent

The Monte Carlo agent evaluates each legal action by simulating N independent game trajectories from that action and averaging the resulting utility. This approach is motivated by the law of large numbers: the sample mean (expectation) of N independent rollouts converges to the true expected utility of the action as N grows:

$$\mu_a = \frac{1}{N}\sum_{i=1}^{N} G_i \xrightarrow{N\to\infty} E[\,G \mid a\,]$$

where G_i is the utility returned by the i-th rollout from action a. The agent selects the action with the highest sample mean:

$$a^* = \arg\max_{a\in A(s)} \mu_a$$

Each rollout applies the candidate action, then simulates forward for a fixed horizon of $H = 15$ turns. Both the agent and all opponents follow a uniform random policy within the rollout. This keeps the rollout computationally cheap and ensures that the Monte Carlo estimate reflects unbiased random play rather than heuristic-based play.

The total computational cost per decision is $O(|A(s)|\cdot N\cdot H)$, and the rollout budget is distributed uniformly across all actions regardless of their estimated value. This uniform allocation is the primary difference from the MCTS agent described in the following subsection.

The utility function used to evaluate rollout outcomes is:

$$u = \pm 1_{\text{win}} + w_{vp}\cdot \Delta\text{VP} + w_\pi \cdot \pi + w_\varepsilon \cdot \varepsilon$$

where $\pm 1_{\text{win}}$ is $+1$ for a win and -1 for a loss in terminal states and 0 otherwise, ΔVP is the victory point lead over the strongest opponent, π is a production score measuring expected resource income per turn from settlements and cities:

$$\pi = \sum_{v\in\text{owned}} (1 + \mathbf{1}_{\text{city}}) \sum_{h\in N(v)} p_h$$

and ε is the count of reachable empty settlement spots, measuring board expansion potential. For the Monte Carlo agent, weights are set to $w_{vp} = 0.25$, $w_\pi = 0.05$, and $w_\varepsilon = 0.02$. The conservative w_{vp} reflects the high variance inherent in purely random rollouts: a random future is a noisy estimate of true action value, and weighting the victory point signal too heavily would amplify that noise into a potentially poor action selection. This can be equilibrated within reason with a higher rollout and/or horizon count.

2.5 Monte Carlo Tree Search Agent

The Monte Carlo Tree Search (MCTS) agent addresses the primary limitation of flat Monte Carlo: uniform budget allocation. With a fixed total budget of B rollouts, spending equal rollouts on every

action wastes compute on clearly inferior candidates. The MCTS agent instead uses the Upper Confidence Bound (UCB1) algorithm [ACBF02] to allocate rollouts adaptively, concentrating budget on actions that appear promising.

UCB1 selects the action to simulate next by maximizing an upper confidence bound on each action's estimated value:

$$a^* = \arg\max_j \left[\bar{x}_j + C\sqrt{\frac{\ln N}{n_j}} \right], \quad C = \sqrt{2}$$

where $\bar{x}_j = \sum_{g_j}/n_j$ is the running mean utility of action j, g_j is the utility thus far from rollout j, n_j is the number of times action j has been sampled, N is the total number of rollouts across all actions, and C is an exploration constant. The exploitation term $\bar{x}_j$ rewards actions that have performed well; the exploration term $C\sqrt{\ln N/n_j}$ grows when an action has been under-sampled relative to the total budget, ensuring that no action is permanently ignored.

Auer et al. [ACBF02] prove that UCB1 achieves $O(\ln N)$ cumulative regret, meaning the total loss from not always selecting the optimal action grows only logarithmically with the number of rollouts. This contrasts with flat Monte Carlo, whose uniform allocation incurs linear regret by continuing to sample inferior actions at the same rate as superior ones.

The MCTS agent operates in two phases. In the seed phase, one rollout is performed for each legal action to initialize statistics. In the UCB1 phase, the remaining budget $\max(0, B - |A(s)|)$ is allocated iteratively by selecting the action with the highest UCB1 score, running one rollout, and updating that action's mean and visit count. The total budget is fixed at $B = 300$ rollouts per decision, independent of the number of legal actions. This ensures consistent computation regardless of action space size, which varies considerably between the placement phase (up to thirty-five candidate vertices) and mid-game decisions (typically ten to fifteen actions).

The utility function is identical in structure to that of the Monte Carlo agent but uses $w_{vp} = 0.50$ rather than 0.25. This asymmetry is implemented since heuristic-guided rollouts produce lower variance utility estimates than purely random rollouts.

It is important to note that this implementation applies UCB1 only at the root node, selecting among immediate legal actions from the current state. Full Upper Confidence Trees (UCT) [KS06] extends the tree recursively, applying UCB1 at every internal node and expanding the tree one node per iteration.

3 Results

Agent performance was evaluated across two tournament formats: a round-robin 1v1 format in which each agent pair played fifty games, and a four-player format in which all four agents competed simultaneously across forty-seven games. Performance is reported by win rate and mean final victory points.

3.1 1v1 Results

Figure 3 summarizes win rates and mean final victory points across all 1v1 matchups. The most prominent finding is the consistent dominance of the heuristic agent over both search-based agents. The heuristic won 80% of games against Monte Carlo and 76% of games against MCTS, despite both search agents employing lookahead mechanisms that the heuristic entirely lacks. This result suggests that under the rollout budget constraints imposed in this study, utility functions based on domain knowledge are a more reliable guide to action selection than simulation-based evaluation.

Mean final victory points follow a consistent hierarchy across all matchups: random agents finish with a mean of 2.3 VP, Monte Carlo agents with 6.3 VP, MCTS agents with 7.0 VP, and heuristic agents with 9.9 VP. This result holds regardless of opponent, suggesting that the performances are attributable to agent strategies rather than matchup-specific factors. The heuristic agent's mean of 9.9 VP is near the 10 VP win threshold, indicating that it consistently reaches the endgame in a competitive position even when it does not win outright.

The direct comparison between the two search agents shows MCTS winning 58% of games against Monte Carlo. This marginal edge is consistent with the theoretical advantage of UCB1-based budget

allocation over uniform allocation: by concentrating rollouts on promising actions and using heuristic-guided simulation for the full rollout horizon, MCTS produces lower-variance utility estimates than flat Monte Carlo, which samples all actions equally under a purely random rollout policy. However, the gap remains modest under the budget constraints of this study, with both agents limited to a rollout horizon of fifteen turns and Monte Carlo limited to fifteen rollouts per action. These constraints bound the quality of simulation-based evaluation for both agents and partially explain why neither closes the gap with the heuristic in 1v1 play.

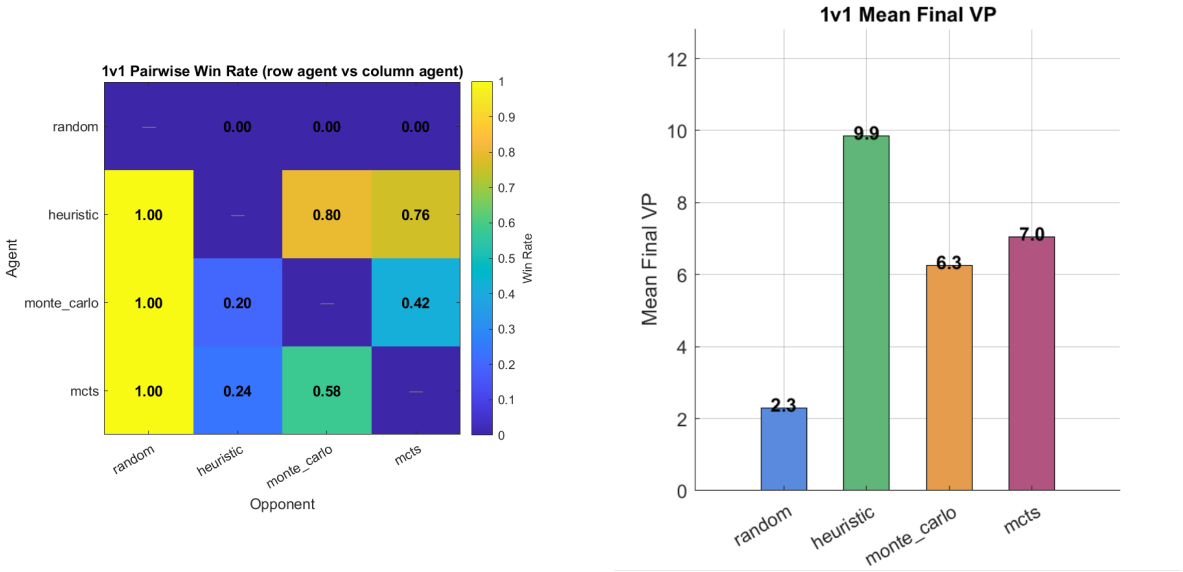

Figure 3: Left: 1v1 pairwise win rate heatmap. Each cell reports the win rate of the row agent against the column agent across fifty games. Blank cells indicate matchups that were not run. Right: mean final victory points by agent across all 1v1 appearances regardless of outcome.

3.2 4-Player Results

As shown in Figure 4, the heuristic agent won 48.9% of the forty-seven simulated games, nearly double the 25% baseline expected from a uniform random competitor in a four-agent field. This is the notable result of the four-player format.

MCTS won 29.8% of four-player games, and Monte Carlo won 21.3%, while the random agent won 0% of games. The ordering of MCTS above Monte Carlo in the four-player format is consistent with the 1v1 result, but the gap between them widens relative to 1v1 play. In the 1v1 format, MCTS held a 58% to 42% edge over Monte Carlo; in the four-player format, the margin increases to 29.8% versus 21.3%. This divergence suggests that the adaptive budget allocation provided by UCB1 becomes relatively more valuable as the number of competing agents grows and the action space becomes more complex. With more opponents producing resources and occupying vertices, the set of consequential actions narrows, and the ability to concentrate rollout budget on those actions confers a greater advantage.

Mean final victory points in the four-player format reinforce these findings. The heuristic finishes with a mean of 8.4 VP, consistent with its 1v1 performance. 6.3 VP, and the random agent with 2.6 VP.

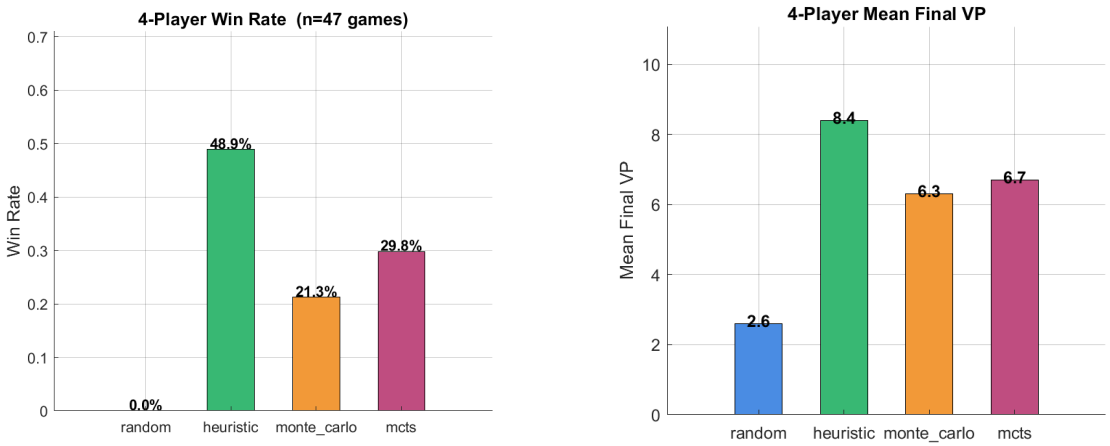

Figure 4: Left: 4-player win rate by agent across forty-seven games. The dashed line indicates the 25% chance baseline for a four-agent field. Right: mean final victory points by agent across all 4-player appearances regardless of outcome.

4 Discussion

The results of this study reveal a consistent performance hierarchy across agent strategies that holds in both tournament formats. The heuristic agent dominates in all conditions, the two search-based agents occupy an intermediate tier, and the random agent provides a stable lower bound. This section interprets these findings, discusses the limitations of the current implementation, and identifies directions for future work.

Computational constraints and search agent limitations. The Monte Carlo and MCTS agents were evaluated under a rollout horizon of fifteen turns and a rollout breadth of fifteen simulations per action for Monte Carlo. These constraints were imposed by available computing resources and represent a meaningful limitation on the quality of simulation-based evaluation. Under a shallow horizon, rollouts terminate before many consequential game events occur, and the utility signal returned is dominated by the current game state rather than the long-term consequences of the candidate action. Furthermore, the reduced rollout breadth could have resulted in many potential moves going unseen. The heuristic agent is not subject to this limitation: it evaluates actions from the current state, and its scoring functions reflect knowledge that remains valid regardless of how far the game has progressed. The dominance of the heuristic over both search agents is therefore best understood not as a failure of search-based methods in general, but as a consequence of the specific budget regime under which they were evaluated. Whether increasing rollout breadth and depth would close the gap between search agents and the heuristic remains an open question that future experiments can address.

Agent strategies and improvement paths. Each agent in this study represents a distinct computational philosophy, and each has a natural improvement path. The random agent serves purely as a baseline and has no meaningful improvement path by design. Its role is to establish a floor against which the contribution of any decision-making strategy can be measured.

The heuristic agent's strength is in its direct use of state knowledge to score functions that are fast, interpretable, and effective under constrained compute. Its primary limitation is that its weights are hand-tuned rather than learned from experience. A natural improvement would be to treat the weight vector $(w_p, w_n, w_d, w_b, w_r, w_c)$ as a parameter to be optimized, using evolutionary algorithms or gradient-based methods to tune weights against a self-play objective. This would transform the heuristic from a fixed rule follower into an adaptive strategy that improves with experience.

The Monte Carlo agent's primary limitation is its uniform budget allocation and purely random rollout policy. Both can be addressed independently. Replacing uniform allocation with a smarter sampling strategy. For example, pruning clearly inferior actions after an initial screening phase would reduce wasted compute. Replacing the random rollout policy with a lightweight heuristic would increase the signal quality of each simulation. These are changes that the MCTS agents implements.

The MCTS agent's most significant improvement path is the extension from flat UCB1 to full UCT

[KS06]. This would allow the agent to reason about multi-step consequences rather than evaluating only immediate actions. A further extension, motivated by the success of AlphaZero in chess and Go, would replace heuristic-guided rollouts with a learned value function trained on self-play data. Rather than simulating fifteen turns of heuristic play to estimate action value, a neural network could evaluate the post-action state, providing a faster and potentially more accurate signal. These extensions would substantially increase computational cost but represent the natural trajectory of search-based agent development in this environment.

The gap between agent play and human play. None of the agents implemented in this study engage in agent-to-agent trading, bluffing, or alliance formation. In standard Catan, human players negotiate resource trades bilaterally, misrepresent their strategic intentions to avoid being targeted by the robber, and form informal alliances to suppress the current leader. These behaviors introduce a cooperative-competitive dynamic that is absent from the present model. The current agents solve a parallel optimization problem in which each player independently maximizes its own resource accumulation and building prowess respective to the other agents. Real Catan is a repeated negotiation game in which value is created through exchange and then competed over, and in which reputation, threat, and coalition dynamics shape outcomes in ways that no individual optimization can capture.

Outlook. Several directions for future work follow naturally from this study. The most immediate is a systematic investigation of how increasing rollout breadth and depth affects the relative performance of Monte Carlo and MCTS against the heuristic. The current study cannot distinguish between the hypothesis that search-based methods are fundamentally inferior to heuristic methods in this environment and the hypothesis that they are simply underpowered under the budget constraints used here. Running experiments across a range of rollout budgets would resolve this question directly.

A second direction is the evaluation of agents against human players. The simulation environment developed for this study could be ported to a Python implementation capable of interfacing with an online Catan platform, enabling the heuristic and MCTS agents to compete in real games against human opponents. Such experiments would also generate data on how human players respond to non-human opponents, which is itself a question of interest in the study of human-agent interaction.

5 Acknowledgements

The author would like to thank Dr. Benjamin Seibold for his guidance throughout this project, for his feedback on the project proposal and midterm report, and for his instruction in mathematical modeling and simulation over the course of the semester.

The following references were central to the theoretical foundations of this work. Auer, Cesa-Bianchi, and Fischer [ACBF02] provided the mathematical basis for the UCB1 algorithm and its regret guarantees. Kocsis and Szepesvári [KS06] established the UCT framework that motivates the MCTS agent architecture. Swiéchowski et al. [SPMK15] provided broader context for heuristic and Monte Carlo methods in general game-playing agents.

Artificial intelligence assistance was used in the development of two components of this project. The game engine, `catan_core.m`, is an approximately 1900-line MATLAB file that implements the complete Catan ruleset, live visualization, and a tournament runner. All agent logic in `agent_random.m`, `agent_heuristic.m`, `agent_montecarlo.m`, and `agent_mcts.m` were designed, implemented, and verified by the author.

References

[ACBF02] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[Kla24] Klaus Teuber. Catan official website. https://www.catan.com/, 2024. Accessed: May 4, 2026.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, pages 282–293. Springer, 2006.

[SPMK15] Maciej Świechowski, Hyunsoo Park, Jacek Mańdziuk, and Kyung-Joong Kim. Recent advances in general game playing. *The Scientific World Journal*, 2015:986262, 2015.